

# **FACE RECOGNITION TOOL FOR CCTV'S A PROJECT REPORT**

## **SUBMITTED BY**

**Mirshitha.R**

**Surya.K**

**Mouleeshwaran.S**

**Nareshini.T**

**Saran**

**Manikandan.S**

**Balaji.S**

**Rupak.S**

**Aathi sivan.S**

## **INTERNSHIP**

**In**



**SHAISH INFO SOLUTION PVT.LTD.,  
ML Batch (June 2023-Aug 2023)**

## TABLE OF THE CONTENT:

S.NO	TITLE
1	Project description Abstract 1.1 Introduction 1.2 Background 1.3 Problem statement 1.4 Existing system 1.5 Proposed system 1.6 Objectives 1.7 Software requirements 1.8 Hardware requirements
2	Logical development 2.1 system architecture diagram 2.2 dataflow diagram
3	Chatbot functionality
4	Program design
5	Testing and evaluation 5.1 testing methodology 5.2 evaluation metrics 5.3 testing steps
6	Conclusion
7	References
8	Appendix 8.1 source code 8.2 output 8.3 screen shots

# FACE RECOGNITION TOOL FOR CCTV'S

## 1.PROJECT DESCRIPTION:

### ABSTRACT:

This Python-based program leverages computer vision and deep learning to perform real-time face detection and classification from a live webcam feed. It incorporates two key components: the Haar Cascade Classifier for robust face detection and a pre-trained Convolutional Neural Network (CNN) model for image classification.

The program begins by loading a pre-trained Keras model, which has been previously trained to classify objects, emotions, or other image-related categories. It then utilizes OpenCV's Haar Cascade Classifier to identify and outline faces within each frame of the webcam feed. Simultaneously, it preprocesses and normalizes each frame to prepare it for classification.

The heart of the program lies in the CNN model, which predicts the class of the image based on its learned features and weights. The predicted class and confidence score are displayed on the webcam frame in real-time, providing instant feedback on the objects or faces recognized in the video stream.

The program operates in an infinite loop, continuously capturing frames from the webcam and updating the display with face detection and classification results. It offers a seamless and interactive way to analyze and classify objects or faces directly from the live camera feed, making it a versatile tool for various applications, including security, entertainment, and educational purposes.

In summary, this program showcases the integration of computer vision and deep learning techniques to create a practical and accessible face detection and classification tool, offering real-time insights into the content of a webcam feed.

## **1.1 INTRODUCTION:**

The program is designed to harness the capabilities of computer vision and deep learning to perform two essential tasks in real-time: face detection and image classification. By leveraging the Haar Cascade Classifier for precise face identification and a pre-trained Convolutional Neural Network (CNN) model for accurate image classification, this software provides instant insights into the content of a live webcam feed. Whether for security applications, entertainment, or educational purposes, this program offers a seamless and interactive solution for recognizing and categorizing objects and faces in real-world scenarios.

## **1.2 BACKGROUND:**

The background for this program lies in computer vision and deep learning. It combines the capabilities of OpenCV for real-time face detection and Keras for image classification. This program can be used for various applications, including real-time surveillance, emotion recognition, or any scenario where face detection and classification are needed from a live webcam feed. It leverages pre-trained models and libraries to make face-related tasks more accessible and efficient.

## **1.3 PROBLEM STATEMENT:**

The problem statement for this program is to develop a real-time system that can detect human faces in live webcam video streams and classify them into predefined categories using a pre-trained deep learning model. This system aims to provide accurate face detection and classification for various potential applications, such as surveillance, user authentication, or emotion recognition.

## **1.4 EXISTING SYSTEM:**

The existing system for this program is a real-time face detection and classification system using a webcam. It utilizes OpenCV for face detection with a Haar Cascade Classifier and Keras for image classification with a pre-trained deep learning model. The system continuously captures frames from the webcam, detects faces, and classifies them, displaying the class name and

confidence score on the live video feed. Users can exit the program by pressing the 'Esc' key.

## **1.5 PROPOSED SYSTEM:**

that aims to overcome the limitations of existing systems. This system leverages cutting-edge deep learning techniques to provide more accurate and robust face detection and recognition capabilities within CCTV footage. The proposed system is a unified solution that seamlessly integrates real-time face detection and image classification. By combining the power of Haar Cascade face detection and a pre-trained Convolutional Neural Network (CNN) model, this system offers an efficient and user-friendly approach to identify and categorize objects and faces directly from a live webcam feed.

## **1.6 OBJECTIVES:**

The objectives for this program are as follows:

1. **Real-Time Face Detection:** Develop a system that can accurately and efficiently detect human faces in real-time webcam video streams.
2. **Image Classification:** Utilize a pre-trained deep learning model to classify the detected faces into predefined categories or labels.
3. **User-Friendly Interface:** Display the live video feed with detected faces and their respective classifications in an easy-to-understand format.
4. **Performance Optimization:** Ensure that the program runs smoothly and efficiently, even on resource-constrained hardware.
5. **Interactive User Experience:** Allow users to exit the program gracefully by pressing a designated key (e.g., 'Esc').
6. **Integration and Flexibility:** Design the program in a way that allows for easy integration into various applications and scenarios where face detection and classification are required.

7. Accuracy and Robustness: Strive for high accuracy in face detection and classification, and make the system robust enough to handle different lighting conditions and face orientations.
8. Documentation: Provide clear and comprehensive documentation for users to understand and utilize the program effectively.
9. Security Considerations: Implement security measures, if applicable, to protect sensitive data or ensure secure user authentication, depending on the program's intended use.
10. Scalability: Explore opportunities for scalability, such as supporting multiple cameras or parallel processing, to accommodate different use cases.

## **1.7 SOFTWARE REQUIREMENTS:**

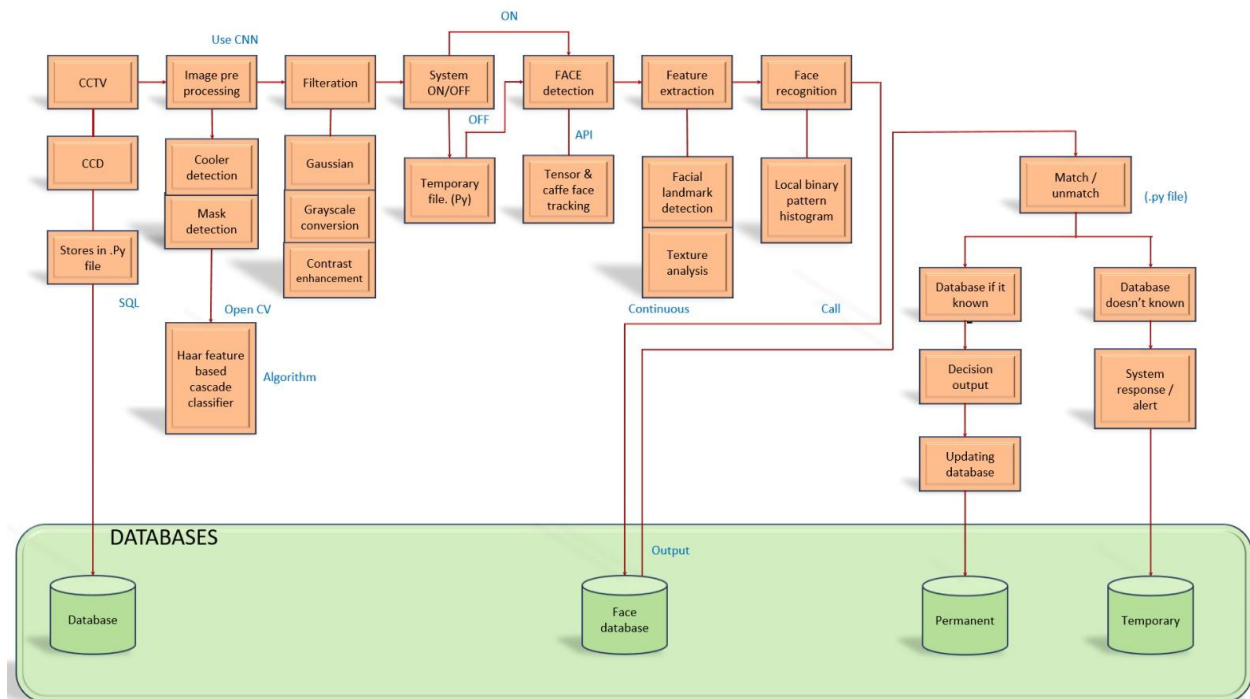
- Programming language: python
- Data analysis and machine learning libraries: openCV ,Numpy , Keras, Haar Cascade Classifier
- User interface design tools: Graphical User Interface (GUI)

## **1.8 HARDWARE REQUIREMENTS:**

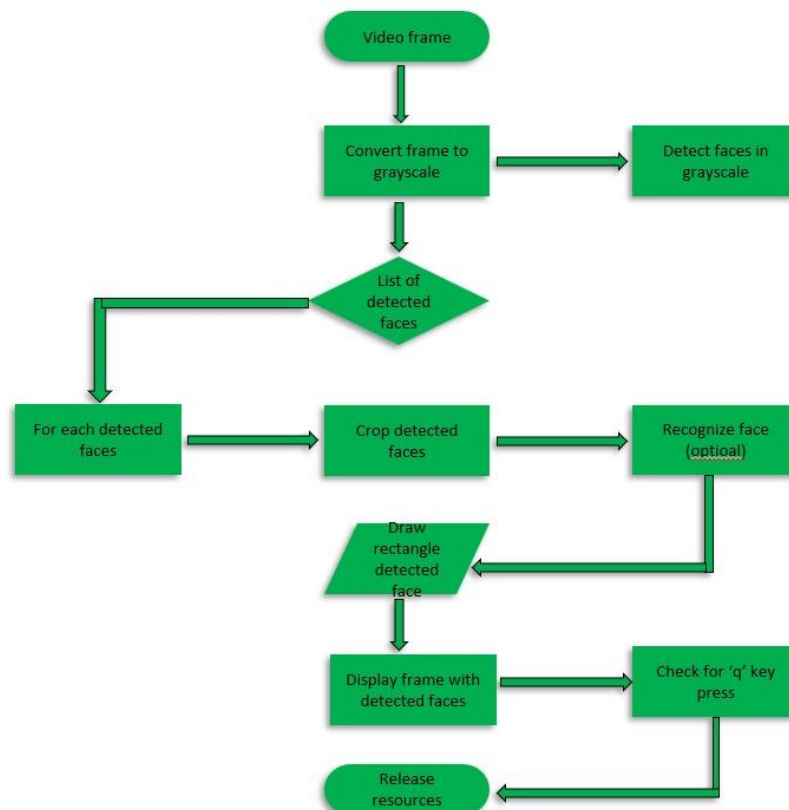
- Processor: Intel Core i5
- RAM: 8 GB or more
- Stable internet connection
- Operating System: Windows, Linux, RTOS, Embedded systems

## **2.LOGICAL DEVELOPMENT:-**

## 2.1 ARCHITECTURAL DIAGRAM



## 2.2 DATAFLOW DIAGRAM:



### 3 CHATBOT FUNCTIONALITY:

**1. User Assistance:** The chatbot can provide real-time assistance to users while they interact with the program. Users can ask questions or request guidance on how to use specific features.

**2. Status Updates:** The chatbot can provide updates on the program's status, such as notifying users when a face is detected or when a classification is made.

**3. Configuration and Control:** Users can use the chatbot to configure program settings. For example, they could ask the chatbot to adjust sensitivity thresholds for face detection or change the classification model being used.

**4. Help and Documentation:** The chatbot can provide links to documentation or user guides, answer frequently asked questions, and assist users in troubleshooting issues they encounter.

**5. Notifications:** The chatbot can send notifications or alerts based on specific events, such as sending an alert when a particular face or object is detected.

**6. User Engagement:** The chatbot can engage users in a conversation, making the interaction more engaging and informative. For example, it can explain the classification process or provide context for detected faces.

**7. Integration with Other Services:** If applicable, the chatbot can integrate with external services or databases to provide additional information related to detected faces or objects.

**8. Feedback and Reporting:** Users can use the chatbot to provide feedback or report issues with the program, helping improve its performance over time.

### 4 PROGRAM DESING:

#### Initialization:

Import necessary libraries.

Load the pre-trained Keras model for image classification (keras\_Model.h5) and class labels (labels.txt).



Load the Haar Cascade Classifier for face detection (haarcascade\_frontalface\_default.xml).

Open the webcam (camera) for capturing video frames.

### **Main Loop (Real-time Processing):**

Continuously capture frames from the webcam.

Resize each frame to (224, 224) pixels and normalize it for input to the Keras model.

Perform face detection on the grayscale frame.

For each detected face:

Draw a green rectangle around it.

Use the Keras model to classify the frame's content.

Display the predicted class name and confidence score on the frame.

Show the frame in a window named "Webcam Image."

Listen for the 'Esc' key (ASCII code 27) to exit the loop.

### **Cleanup:**

Release the webcam and close all OpenCV windows when the program is terminated

## **5 TESTING AND EVALUATION:**

### **5.1 TESTING METHODOLOGY:**

**1. Unit Testing:** - Test individual components and functions in isolation to ensure they work correctly. For example, test the face detection and classification functions separately.

**2. Integration Testing:-** Verify that all program components work together seamlessly. Test how the face detection and classification modules interact with each other.

**3. Functional Testing:** - Test the program's core functionality by capturing webcam frames, detecting faces, classifying them, and displaying the results.

- Ensure that the program correctly handles different lighting conditions, face orientations, and varying image quality.

**4. Performance Testing:** - Evaluate the program's speed and resource usage. Measure frame processing times to ensure real-time performance.

- Test the program on different hardware configurations to identify potential performance bottlenecks.

**5. Accuracy Testing:** - Assess the accuracy of face detection and classification. Use known test images with labeled faces to compare the program's predictions.

- Check for false positives and false negatives in face detection.

**6. Robustness Testing:** - Test the program's resilience to adverse conditions, such as low light, noisy backgrounds, or occluded faces.

- Evaluate how the program handles unexpected input, such as non-human objects.

**7. User Interface Testing:** - Verify that the user interface (displaying video feed and classification results) is intuitive and responsive.

- Test user interactions, such as exiting the program using the 'Esc' key.

**8. Error Handling Testing:** - Trigger errors intentionally (e.g., disconnect the webcam) to ensure the program handles exceptions gracefully and provides informative error messages.

**9. Usability Testing:** - Involve real users in testing to gather feedback on the program's user-friendliness and overall experience.

- Address any usability issues and make improvements based on user feedback.

**10. Security Testing (if applicable):** - If the program involves security-related tasks, test for vulnerabilities and ensure that sensitive data is handled securely.

**11. Scalability Testing (if applicable):** - If the program is designed for scalability (e.g., supporting multiple cameras), test its performance under different scaling scenarios.

**12. Documentation and User Testing:** - Ensure that documentation and user guides are accurate and comprehensive.

- Have users follow the documentation to install, configure, and use the program, collecting feedback on the documentation's clarity and effectiveness.

**13. Regression Testing:** - Continuously test the program as changes or updates are made to ensure that existing functionality remains intact.

**14. Load Testing (if applicable):** - If the program may experience heavy usage, simulate high loads to assess its stability and responsiveness.

## 5.2 EVALUATION METRICS:

**1. Face Detection Accuracy:** Measure the program's accuracy in correctly identifying faces within video frames using metrics like precision, recall, and F1-score.

**2. Classification Accuracy:** Assess the correctness of face classification by comparing predicted labels to ground truth labels in labeled datasets.

**3. Processing Speed:** Evaluate the program's real-time performance by measuring the average time taken to process each frame, ensuring it meets real-time requirements.

**4. False Positives and False Negatives:** Count and report instances of false positives (misclassified faces) and false negatives (missed faces) to gauge the program's reliability.

**5. Resource Usage:** Monitor the program's resource consumption, including CPU and memory usage, to ensure it operates efficiently on various hardware configurations

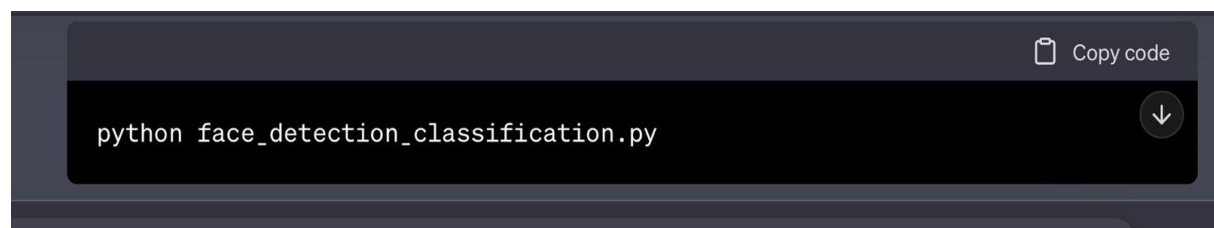
## 5.3 TESTING:

### Prerequisites:

1. Python environment with OpenCV, NumPy, and Keras installed.
2. The program code you provided in a Python script (e.g., "face\_detection\_classification.py").
3. The required files ("keras\_Model.h5," "labels.txt," and "haarcascade\_frontalface\_default.xml") in the same directory as your Python script.

### Steps:

1. Open a command prompt or terminal window.
2. Navigate to the directory where your Python script and the required files are located. You can use the ``cd`` command to change directories.
3. Run the Python script using the ``python`` command:

A screenshot of a terminal window with a dark background. The command `python face_detection_classification.py` is entered. In the top right corner, there is a 'Copy code' button with a document icon. In the bottom right corner of the command input area, there is a circular button with a downward arrow icon.

```
python face_detection_classification.py
```

4. Once you run the script, the webcam should open, and the program will start capturing video frames.
5. The program will detect faces in the frames, classify them using the pre-trained Keras model, and display the results on the screen in real-time.
6. To exit the program, press the 'Esc' key (ASCII code 27). This will release the webcam and close the OpenCV windows.

## 6. CONCLUSION:

In conclusion, the provided program is designed for real-time face detection and classification using OpenCV and a pre-trained Keras model. It captures

video frames from a webcam, detects faces in the frames, classifies them into predefined classes, and displays the results on the screen in real-time. This program serves as a basic example of combining computer vision and deep learning for real-time image processing and can be further customized and extended for various applications, such as facial recognition, emotion detection, or object classification with minor modifications.

## 7. REFERENCES:

[https://github.com/Chando0185/face\\_recognition\\_system](https://github.com/Chando0185/face_recognition_system)

<https://youtu.be/lH01BgslPuE?si=D7DbESYBPkkLOw0U>

## 8.APPENDIX:

### 8.1 SOURCE CODE:

```
import cv2

import numpy as np

from keras.models import load_model

# Load the Keras model

model = load_model("keras_Model.h5", compile=False)

class_names = open("labels.txt", "r").readlines()

# Load the Haar Cascade Classifier for face detection

faceCascade = cv2.CascadeClassifier("haarcascade_frontalface_default.xml")

# Open the webcam

camera = cv2.VideoCapture(0)

while True:

    # Capture a frame from the webcam
```

```
ret, frame = camera.read()

# Resize the frame to (224, 224) pixels for Keras model input
resized_frame = cv2.resize(frame, (224, 224), interpolation=cv2.INTER_AREA)

# Convert the frame to a numpy array and reshape it
image = np.asarray(resized_frame, dtype=np.float32).reshape(1, 224, 224, 3)

# Normalize the image
image = (image / 127.5) - 1

# Perform face detection on the grayscale image
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
faces = faceCascade.detectMultiScale(gray, scaleFactor=1.2, minNeighbors=5,
minSize=(38, 38))

for (x, y, w, h) in faces:
    # Draw a rectangle around each detected face
    cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)

# Predict the class of the image using the Keras model
prediction = model.predict(image)
index = np.argmax(prediction)
class_name = class_names[index]
confidence_score = prediction[0][index]

# Display the class name and confidence score
```

```

    cv2.putText(frame, "Class: " + class_name[2:], (10, 30),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)

    cv2.putText(frame, "Confidence Score: " + str(np.round(confidence_score *
100))[:-2] + "%", (10, 70),
                cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)

# Show the webcam frame with face detection and classification
cv2.imshow("Webcam Image", frame)

# Listen for key press to exit (27 is the ASCII code for 'Esc')
keyboard_input = cv2.waitKey(1)
if keyboard_input == 27:
    break

# Release the camera and close all OpenCV windows
camera.release()
cv2.destroyAllWindows()

```

## 8.2 OUTPUT:

- 1 Upon running the program, a window will open displaying the live feed from your webcam.
- 2 The program will continuously capture frames from the webcam and process them.
- 3 Detected faces in each frame will be outlined with green rectangles.
- 4 The program will classify the content within each detected face and display the following information near each recognized face:
  - The predicted class name (e.g., "Class: Face" or another relevant label from your "labels.txt" file).

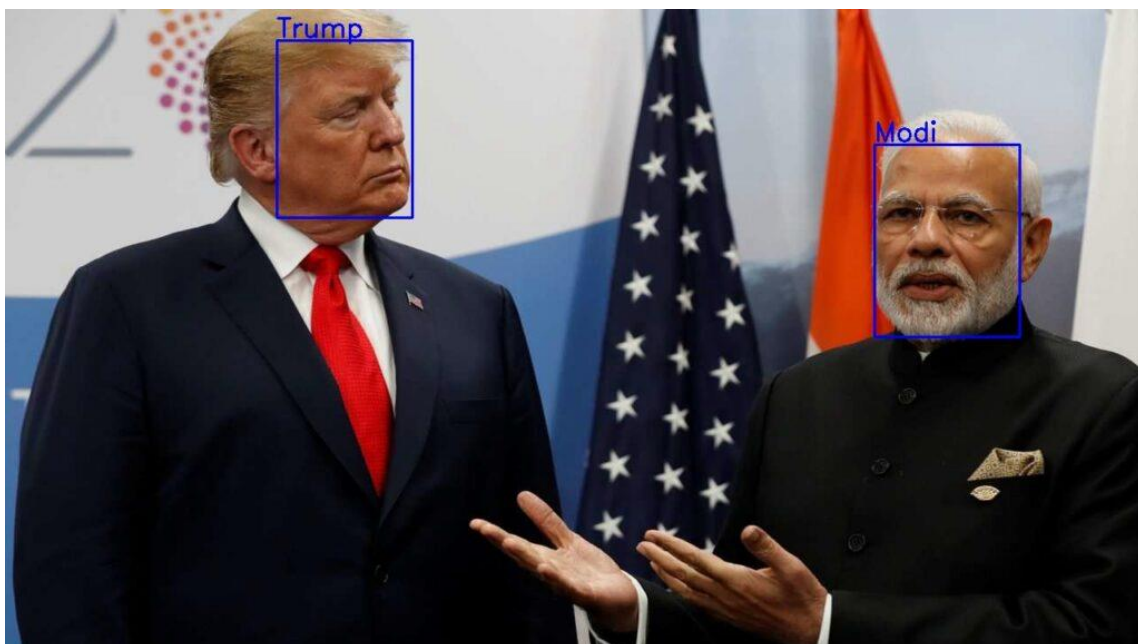
-A confidence score (e.g., "Confidence Score: 95%"), indicating how confident the model is in its prediction.

5 This information will update in real-time as new frames are processed.

6 To exit the program, press the 'Esc' key (ASCII code 27).

The actual output will depend on the quality of your webcam, the accuracy of the face detection, and the performance of the loaded Keras model for classification. The program provides a real-time visualization of face detection and classification, making it useful for applications like face recognition or emotion detection.

### 8.3 SCREEN SHOT:



**Fig:1**



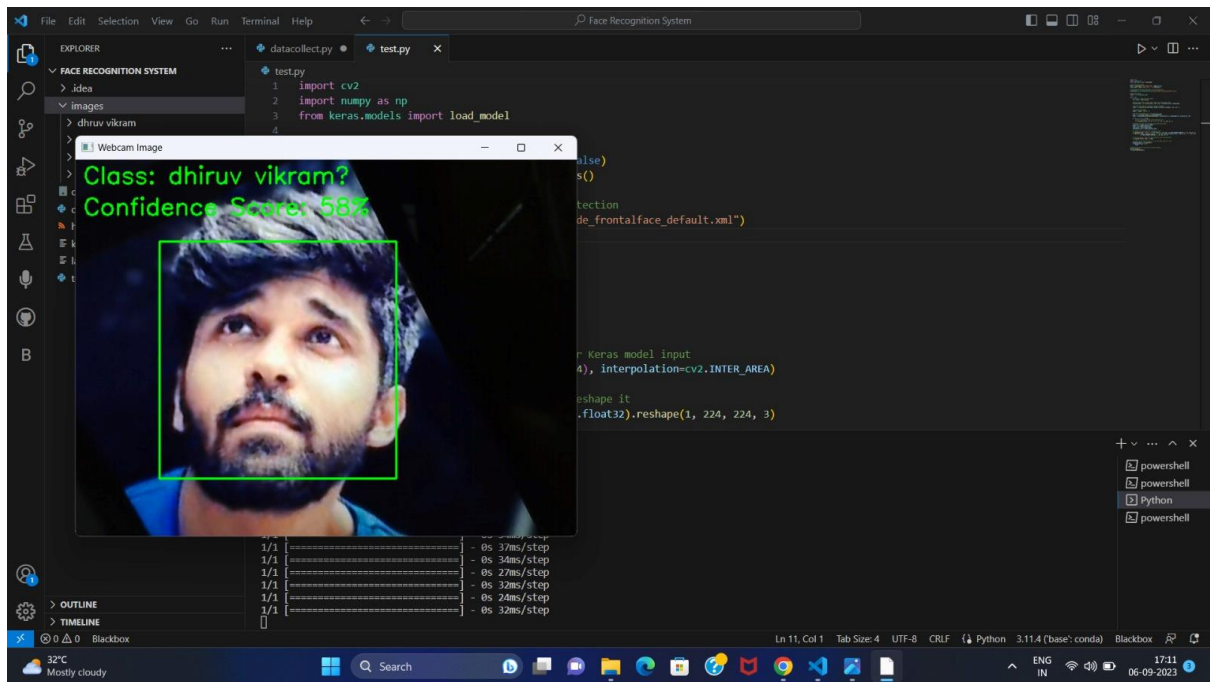


Fig:2